Audio-Visual Resynchronization

Skyler Heininger, Gian Cercena, Nathan Blanchard

December 13, 2024

Abstract

The synchronization of audio and video is crucial for video quality and viewer experience. Desynchronization can be caused in numerous ways. This project introduces a method to detect and measure the audio-visual desynchronization within videos featuring speaking individuals. Using the AVSpeech dataset, we preprocess and desync videos by up to +-1 second. Various sets of audio and visual features are extracted and utilized in different neural networks to produce a regression output of the number of frames the audio is desynced by. Experiments show that using optical flow as the visual feature set greatly improves performance giving a root mean squared error (rMSE) of 4.04 frames, beating the no-information targets. Despite computational limitations that prevented full model convergence, our model demonstrates strong potential for widely generalizable audio-video resynchronization.

1 Introduction

Either during transmission of video, or the physical degradation of the medium it is stored in, the audio of a given video can become desynced. A desynced video is one where the audio and the video do not align, with one leading the other by potentially large amounts. When a video is desynced in such a way, the viewer's experience is disturbed, and the contents can be more difficult to make out.

The issue our project attempts to handle is the detection of desynchronization between video and audio in videos where an individual's face is shown, and more specifically, the amount it is desynced by. This is an important problem to solve for various reasons. From families restoring old home videos, to museums restoring large amounts of old footage, instead of having an individual sit through and adjust each video, massive amounts of time can be saved by shifting this problem over to an automated system. Potential improvements could even lead to systems such as these happening in near-real-time, allowing for even another space of improvement.

This problem has been attempted before in various ways [1, 2, 3]. Ours follows related avenues. Our basic approach is to take video footage of individuals talking, with their face in the video and desynchronize them randomly within a certain range. For each frame in this altered video, we collect audio and visual features, allowing the model to pick up on this problem. This way of solving this issue is ideal since it avoids having to incorporate other models, such as speech detection, and it would work regardless of spoken language, simply relying on the features provided to the model. These features would be fed into a fusion model that merges the features and then provides a regression output being the amount of frames it believes the audio is desynced by, which could then be utilized to shift the video back into its original form.

2 Problem Definition and Algorithm

2.1 Task Definition

Our problem focuses on the resynchronization of videos and their audio, using solely features extracted from both. Primarily we want to do this using a person speaking, where the audio is their spoken words while videos capture the speaker's facial movements when talking. We begin with a dataset of YouTube videos, each with a person's face prominently within the frame of the video. We operate under the assumption that these videos do not have desynchronization problems.

Our first pre-processing step starts with the cropping of the video using OpenCV's [4] headless model, which performs object detection and cropping of the video to just encapsulate the person's face. We have measures in place to reject videos where OpenCV's model finds no face present. The threshold is a face not being detected concurrently for a certain amount of frames. This gets rid of bad videos for our use case, as well as videos that have faces that may be hard to detect. Without a face, we cannot get the visual features related to it in order to feed our model.

We additionally reject videos where the face moves by too many pixels in a video. While this can occur naturally in videos, this generally happens when there are multiple faces close together in the video, causing the algorithm to sometimes select the wrong face as the target. Since generally one person speaks at a time, this can cause issues where the person's face who is being focused on is not the one speaking. Thus, we reject these videos using a distance measure between faces in consecutive frames.

Subsequently, we desynchronize the video and audio. We shift the audio within the range of +- 1 second, chosen uniformly. Note that we also reduce the framerate of each video down to 15 fps, meaning that the audio could be up to 15 frames, or 1 second off.

Once we have desynced cropped videos, our process diverges into two branches where each extracts features from each frame of all videos. The first looks at audio. For our initial approach, since audio isn't split into frames like video is, we manually divide the audio into chunks that correspond to frames of the video and then get features from each. Using the Python package Librosa, we collect features that are tailored for speech processing, such as MFCC, ZCR, and Mel-Spectrogram features, allowing us to capture essential characteristics of spoken language. This step in audio processing outputs up to 118 features per segment of audio corresponding to a single frame of video. Our next approach, which adopted the WLOS architecture [1], computed just 12 MFCC features by processing the audio using a sampling rate of 1600, hamming window of 304, and a hop length of 152.

The second branch looks at the video. Our initial approach used feature extraction from a pre-trained CNN. The CNN chosen was EfficientNetV2 [5] due to its quick processing, high accuracy, and reputation, which will be noted on later. The model outputs 1280 features per frame, which can be reduced via an autoencoder if decided. We also attempted to use a convolutional autoencoder on each frame of video, to downsize each image to a 16 by 8 by 8 tensor. This was used as visual features for various pipelines.

Our most successful form of visual features was optical flows of the lip region, specifically using the Gunnar Farneback algorithm to calculate the vertical pixel level velocities between frames of video. This was applied to just the lip region of each face in a frame, which we used InsightFace's RetinaFace model to determine the location of the lips in each frame.

The visual and audio features are then concatenated together to create the input to the transformer. The label for each video is the amount of frames the audio is desynced, ranging from the previously stated values of -15 to +15. The transformer uses this input to predict a single desynchronization value.

This is an interesting and important problem due

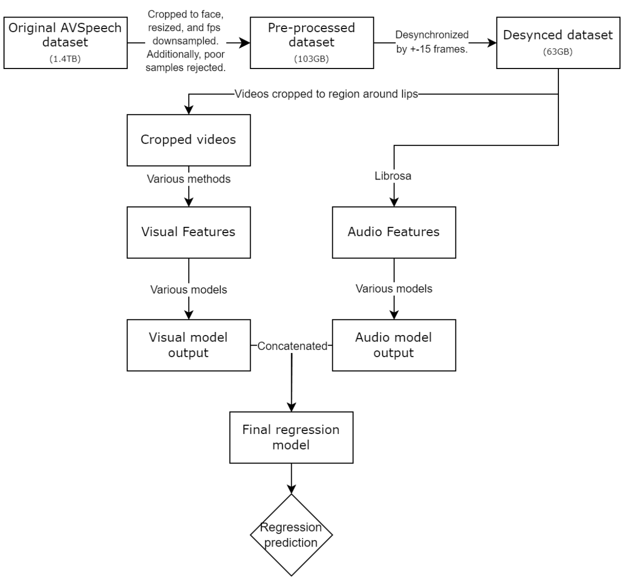

Figure 1: *Pipeline including data processing and models.*

to the fact that a system that uses a comprehensive model such as this could be deployed in an automated, scalable way, allowing for synchronization of video and audio in environments where such media might become desynched. By using deep neural networks such as we intend to in this project, we hope to present a system that can determine desynchronization even in non-perfect environments, for example, where there may be large amounts of background noise, or a lower video quality, all while being robust to other variations such as the language spoken. Lip-synchronization systems like this could be implemented in a large number of areas, such as museum archival restoration projects, or even live streaming services, speeding up tasks, or enhancing user experience, respectively.

2.2 Algorithm Definition

As seen in Figure 1, our model building process follows similar steps for each of the 5 iterations of models we had. We preprocessed our data up to the point where we had our collection of desynced videos noted at the "Desynced dataset" position in Figure 1. The downstream steps were taken several times depending on the architecture.

2.2.1 Model 1 (Transformer Model) Model 1 started with 1024 visual features extracted from a CNN, and 118 features extracted using mel spectrograms and subsequently fusing them together in a transformer model.

These initial 1024 visual features were derived from the model family of EfficientNetV2 (ENV2) [5]. ENV2 was the second iteration of a set of CNNs that attempted to solve the issue of slow training speeds, and low parameter efficiency. Specifically trained and developed in a way to be more efficient than other state-of-the-art models at the time of release, both in time spent training and parameter size, ENV2, such as all other CNNs, outputs a set of visual features procured from an image. The model learns a set of features that it believes is optimal, and produces an output that contains the signal of these features.

These visual features were calculated based on videos cropped to an individual's face, which we thought would give us additional information to assist the model. These features were collected once for each frame and regularized to maintain consistency.

The audio features used were computed using the Python package Librosa [6]. Our first model's feature set includes 13 Mel-frequency cepstral coefficients (MFCCs), their first and second-order derivatives, zero-crossing rate (ZCR), spectral features (centroid, bandwidth, rolloff, and 7 contrast features), root mean square energy (RMS), harmonic-to-noise ratio (HNR), short-time energy (STE), and 64 mel-spectrogram representations. These features were calculated and averaged to create a row of audio data corresponding to each frame of video input. In total, each row of audio data consists of 118 features. These features are tailored for speech processing, capturing essential characteristics of spoken language.

These features were then sent into a transformer model. Transformers are a type of neural network that excel in processing sequential data. They use self-attention mechanisms to weight the significance of different parts of the input sequence, allowing for better understanding [7]. The model took the audio and visual features sets and independently ran them through an attention model in order to learn feature importance. They were then fused, fixed to a set size, sent through a transformer encoder layer, producing an answer ranging from -1 to 1, which was then resized to work with the frame desync range provided.

2.2.2 Model 2 (Autoencoder + LSTM) Our second model utilized a convolutional autoencoder to extract visual features from the data. We trained a convolutional autoencoder on a portion of our dataset, using each frame of each video. The encoder was composed of five convolutional layers, with kernel of size 3, padding of 1, and stride of 2. The decoder was made of another five convolutional transpose layers, recreating the original RGB input image. This portion of the pipeline used a 128 by 128 pixel image of around the mouth region of each face in each frame, however this also included a lot of cheek area, nose, eyes, and chin, likely adding noise to the visual features in general.

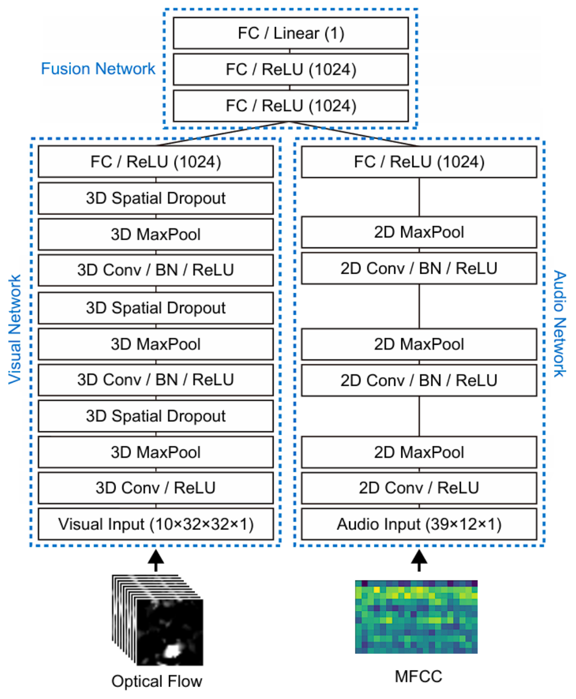

Figure 2: *A diagram of our version of WLOS adapted from Kikuchi and Ozasa [1].*

For this model, the audio features remained the same as our previous model, we used the same 118 features stated above. The visual feature set was then processed through fully connected layers to reduce dimensionality.

Afterwards, they were processed by a long-short-term memory (LSTM) layer for each feature set. LSTM is a variation on RNNs that have been noted to be good at processing sequential data, recognizing long-term dependencies and applying that learned knowledge to its output [7]. The outputs of the LSTMs were concatenated together, and then sent through a fully connected layer to give the regression output.

2.2.3 Model 3 (WLOS) The final model (Figure 2) takes a step away from the previous structures and adopts a structure similar to the WLOS model by Kikuchi and Ozasa [1]. Besides the layer architecture, the chief difference is that visual features were not extracted from some CNN or autoencoder, but instead from optical flow of the lip region.

Optical flow is the per-pixel apparent movement between two frames. This is calculated via the Gunnar Farneback method [8] using OpenCV's implementation (CITE). The face was cropped into specifically the region with the individual's lips using InsightFace's RetinaFace [9] model to detect the location of the lips, which was then resized to images of size 32x32.

The key difference between optical flow and visual features is that optical flow can capture inter-frame dependencies and information, giving the model clearer data about the transitions between each frame. This means the model does not have to pick up on learning that relationship in the data and can divert more of its complexity to solving the regression problem.

The switch to using optical flow was also followed with the reduction of audio features to more closely follow the WLOS architecture. Only 12 audio features were utilized, MFCCs 2-13, removing the first MFCC, all of the additional MFCC derivatives, ZCR, spectral features, RMS, HMR, and STE.

Each of the audio and visual features were again processed independently of each other, utilizing several 3D CNN layers, max pooling layers, and heavy dropout sequences with a rate of 0.5. The processed features were then sent into two fully connected layers to then give the regression output.

The key differences between our model and Kikuchi and Ozasa's was that our video frame rate was different so the 3D CNNs had different input sizes to account for that. We wanted the model to capture the full range of our offset, which while it had fewer frames than the original WLOS, was longer, due to the lowered fps. Accordingly, each visual input was accompanied with 56 MFCC frames compared to their 39.

Their training data was also isolated by individuals, and their train-test split was within individuals in their dataset, meaning they did not generalize to unseen faces. Our approach also used an additional 3D-CNN layer on the visual features and 2D-CNN on the audio features to help capture more relationships within the data.

3 Experimental Evaluation

3.1 Methodology

To evaluate our regression-based method, we used mean squared error (MSE) as our key metric. This

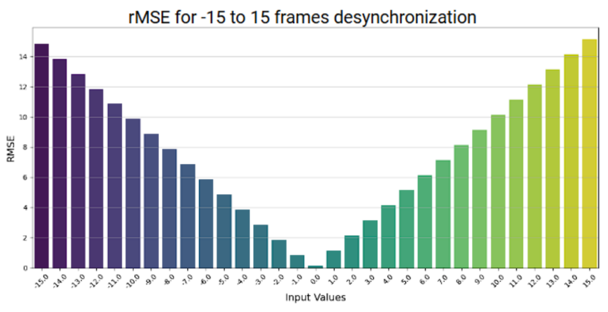

Figure 3: *Average rMSE values for Model 1 with the range of -15 to 15 frames.*

criteria effectively informs our model during training that its numerical outputs are better or worse, stressing egregious errors. This was useful in testing our hypothesis: whether transformers can predict the number of frames difference between a video of an unseen person speaking and their audio.

Our experimental methodology involved using root mean squared error (rMSE) as the loss during training, variance regularization, penalizing the model for making similar predictions. We additionally used a 70-15-15 train-validation-test split in order to evaluate our model on the validation set during training. This split was determined randomly to start, splitting using the ids of YouTube videos in the dataset. Since some videos had multiple clips in the dataset, we were sure to only include a video's clips in a single partition of the train-validation-test split. This is realistic as it avoids data leakage of one video's content from the train set to the test set.

The performance data collected is the rMSE, which evaluates how far off a given guess is in terms of frames. For instance, an rMSE of 3.4 means an average prediction is off by 3.4 frames, with larger errors being weighted more due to the squared nature of the error metric. We additionally collected a histogram of the average rMSE values for each level of desynchronization. Considering we have a range of -15 to 15, our expected value for rMSE is around 8.66, if zero is guessed for every sample. We additionally tried reducing our range of frames from -15 to 15 to -7 to 7, effectively halving our range. The expected rMSE value for this new range is 4.32, given guessing the optimal value of zero.

3.2 Results

Our first two models provided average rMSE values close to what should be possible when predicting zeros. For instance, without regularization, Model 1 predicted the frame offset in the range of -15 to 15, the average rMSE was 9.05, which is worse than simply

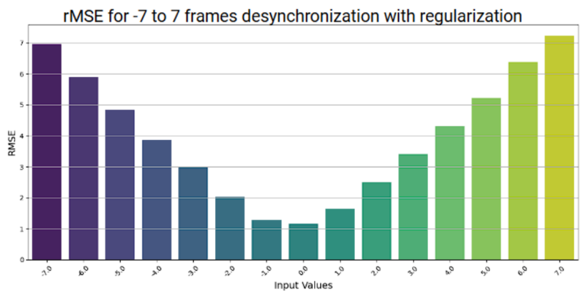

Figure 4: *Average rMSE values for Model 1 with the range of -7 to 7 frames, with minor variance regularization.*

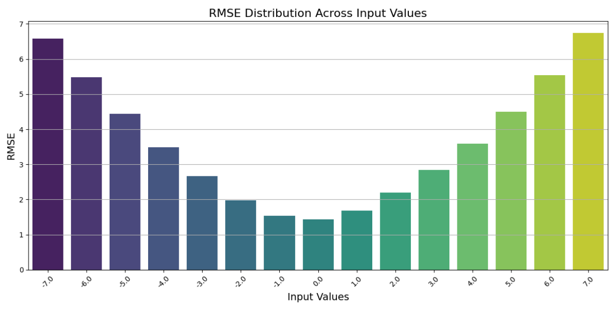

Figure 5: *Average rMSE for deeper WLOS implementation.*

predicting zeros. This model generally predicted zeroes, as is seen in Figure 3, where the average rMSE of a given frame offset is equal to that frame offset.

Adding variance regularization to this helped the model to predict differently than zero, but did not improve average rMSE significantly past predicting all zeros. In fact, the average rMSE for this model was 4.32, being the same as the rMSE for predicting all zeroes.

Figure 4 displays the predictions for this model when using variance regularization and indicates the still-prevalent issue of predicting zeros. Models 2 and 3 (with auto-encoded features) also saw this issue of predicting zeros and no convergence, indicating likely issues with the data.

Model 3 with vertical optical flow visual features displayed convergence and improvement of the average rMSE past predicting all zeros. The average rMSE for this model was 4.04, a clear improvement over that of predicting all zeros, being 4.32. This rMSE still indicates that a guess is on average 4.04 frames off of the correct desynchronization.

Figure 5 indicates the average rMSE per each correct desynchronization (input values). From this graph, it is noteworthy to see that the average rMSE in the range of -7 to -2 and +3 to +7 are all below what would be expected when predicting zero, while the range of -1 to +2 (inclusive) are all above what is expected. This indicates the model has begun to predict things other than zero and has begun to learn important features from the data.

It is important to note that due to the pure amount of data and lengthy training times that accompany this, our models have likely not completely converged. For instance, Figure 6 displays a loss graph from a training run-through, which was performed over 25 percent of the dataset. This indicates the model is still converging, as validation and training losses are decreasing together.

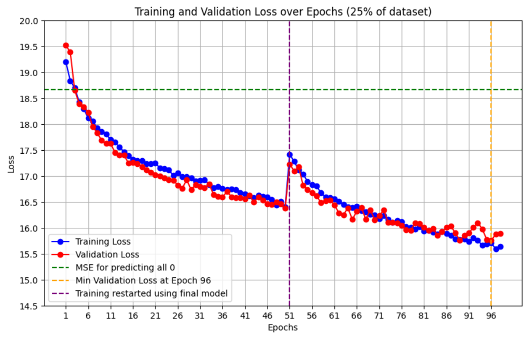

Figure 6: *Loss graph using 25% of the dataset. Training was restarted partway through due to hitting maximum time on VACC.*

The issue here is that these 50 epochs took two days of GPU runtime, meaning that converging completely would likely take a lot more time. Using the entire dataset would take many more resources. Additionally, increasing learning rate did not improve convergence rates, and generally made it harder for the model to converge.

3.3 Discussion

Our hypothesis is slightly supported by our results. Although many models did not perform better than their no-information rates, refinement of our audio features, visual features, and models has led us to creating models that perform better than no-information rates. While these results could be improved with additional modifications to models, features, pretraining, and hyperparameter training, they still show a distinct possibility of this task having generalizability.

Early models revealed issues with how we handled our audio and visual features. Originally, these were concatenated together, which provides little information to models for how they are related. This was iterated upon, and later models correctly provided the audio and visual features into two different parts of models. For instance, our adaptation of the WLOS model provided an audio features stream and visual feature stream which allowed the model to learn each of the streams separately rather than having no clear relation between features.

Our results also provide the importance of the type of training data. The features taken from ENV2 and our convolutional autoencoder provided little actual use. For instance, ENV2 was trained for classification purposes, so while it could accurately provide features allowing a model to understand the structure of a human's face, the features themselves provide little information of the movements of a person's face. The autoencoder contained the information necessary for generally reconstructing a face, but this was unspecific, used a large portion of the face instead of just the lips, and again did not contain information about the movement between frames.

Meanwhile, the optical flow features provided results better than no-information rates when using just the lip region, presumably due to the direct encoding of inter-frame information. There are likely other options for visual features, but this has provided the insight of how specific the features need to be when the dataset itself contains so many different faces, languages, and more. The inherent noise of the dataset made this difficult, since our goal was generalizability. Pretraining on a more-ideal dataset could likely help with this issue.

Our results, including Figure 6, indicate that our models could also benefit from additional training. Since the models have not completely converged, there is the possibility for them to improve. However, following the rate of convergence, it would likely take several hundred epochs to converge to achieving an average rMSE of 1, which is similar to the results found by the WLOS paper [1]. This would correspond with at least weeks of computing time, requiring a GPU, which is resource intensive and costly for VACC partitions that could be used for other projects. We also want to experiment more with deeper models, though this would also make training lengthier and more resource intensive. Our attempts to make these deeper models has so far resulted in long training times and poorer results than our current best model, but there is still a lot of research that we must do into these models as we strongly believe that they have potential to show big improvements with more refinement and resources. We plan to continue training the models to improve results further, even after the completion of this project.

4 Related Work

There are previous attempts to solve this problem using statistical models and neural networks. Kumar et al. (2009) [2] attempts to solve this issue using a bimodal linear prediction model. Their venture to solve this issue focuses on attempting to predict the visual and audio features based on previous time steps. If the features meet certain metrics, they can be considered synchronized. They captured geometric features like the lip height and lip width of the speaker's face, and attempted to measure the statistical dependency between both the audio and visual features.

T. Kikuchi and Y. Ozasa (2018) [1] take another step in this problem by utilizing a regression-based CNN in order to predict the frame offset, which we based Model 4 off of. They used the vertical optical flow calculated from the lip region of the individual's face to capture the movement overtime and sent the information into the CNN in batches of 10 frames. Additionally, they used MFCCs for audio features, capturing the features on a per-frame basis. One extra step they took was to pre-train the model on a simpler problem, a binary classification problem of whether or not a given video was desynched. The models output were good, achieving a frame offset mean absolute error of 0.9.

While we improved upon this model, our errors were not that low. This is due to several reasons. The first is that this paper uses a maximum frame offset of +-9 in 25 fps videos. This results in an output range of 19, and a maximum desync time of 0.36 seconds. Ours looks at +-15 frames in 15 fps videos, increasing our maximum offset time to 1 whole second. Additionally, they use a high-quality dataset that focuses specifically on 3 second clips of several individuals' lips while they speak a sentence. Their models are also trained and tested on the same individuals, meaning their results do not necessarily indicate generalizability, rather the ability for models to work when using this form of data. Our data is much more irregular, with many different individuals and languages being spoken. We additionally tested our models on unseen subjects, rather than stratifying by portions of a video. Our goal was to generalize this model for unseen faces and facial movements. These reasons led our model to have a higher error.

Khosravan et al. (2019) [3] incorporates transformer models within their paper, examining the usefulness of them when it comes to this problem. Their model structure focuses on the attention mechanism to weight certain parts of videos based on their relevance towards determining the frame offset. Their frame offset range was +-130 frames, giving them a range of 261 values, and a time range of 4.3 seconds given a 29.97 Hz frame rate (30 fps). After extracting joint audio-visual features using a CNN across both audio and visual features, they predict the frame offset using a decision layer following an attention module. Their best models gave an error of 28 frames, which significantly improves upon random guessing.

EfficientNetV2 (2021) [5] intends to improve sizeable upon previous CNNs particularly in the training speed and parameter efficiency, leading to smaller, more effective models on a per parameter basis. Specifically, EN looks to improve upon its early version by introducing a new type of convolution to improve speed, Fused-MBConv. This convolution removes slow depthwise convolutions which can be slow in earlier layers, allowing for a large time speed up while still keeping the sophistication of the previous MBConv, allowing for a variety of features to be represented. Additionally, in earlier epochs, the model will train with smaller image sizes and weaker levels of regularization, which it increases over time in order to not get overwhelmed on many details early in the training sequence. While being 11x faster, and upwards of 37% smaller than the previous iteration of EN, state-of-the-art results are still achieved, getting 85% accuracy on ImageNet.

5 Code and Dataset

Dataset: AVSpeech: Audio Visual Speech Dataset (looking-to-listen.github.io) [10]

The AVSpeech dataset consists of 290,000 YouTube videos, containing 4700 hours of content. The videos chiefly consist of Ted Talks and how-to videos that contain a single person speaking, with their face clearly visible. Additionally, this corpus of videos contains many languages, giving a wide variety, helping the model generalize to languages outside of those that predominantly inhabit this online space. The base form of this dataset is not the actual videos, but instead YouTube video ids. Instead of downloading all videos, an academic torrent was found, downloaded to a local device, and then transferred to the VACC. The original data is located on the VACC under the location: "/gpfs2/classes/cs6540/AVSpeech". This folder contains the original dataset and our processed versions of the dataset.

Our GitHub containing all code is here, each folder within the GitHub matches the processing required to get from each step in the dataset directories to the next directory. github.com/njblanch/CS6540-Final-Project.

The following is a tutorial to reproduce our results by running code found on our GitHub repository:

1. Download the zipped files from the AVSpeech dataset.
2. Use a command line utility to extract the zipped files to 2_unzipped/.
3. 3_cropped/process_video_clips.py: This will first downsample videos to a framerate of 15 frames per second. This will increase processing speed and make the size of our data more feasible to process. Next, this will isolate the head of the person speaking in the video, cropping to a bounding box of size 256x256.
4. 4_desynced/desync_audio.py: This script trims the video by 1 second on either end, preserving audio. This audio clip is then placed at a random location on the video, creating a desynchronization of +/- 15 frames, or one second, between the audio and the video. The amount of desync is then appended to the video name and the video is re-saved in 5_audio. Note that this step of processing is limited to video clips that are 6+ seconds long.
5. 5_audio/audio_features.py: This step extracts audio features, saving all feature observations of the same video id to a single CSV to make the train-test split easier to perform without accidentally having training data leak into our test data. To identify audio features that correspond to which video clip, this has rows 'video_id' and 'video_number'. These features include MFCC audio features, commonly used in speech processing tasks. There are multiple rows of audio feature readings per video frame.
6. 6_visual_features/visual_features.py: This performs visual feature extraction for each frame of video using EfficientNetV2, a pre-trained CNN. The model by default outputs 1280 features per frame, with 'video_id', 'clip_number', 'frame_number', and 'desync' specified along with each row of features. The number of features can be reduced via an autoencoder, something we take advantage of for test runs, often setting the visual output feature count to 128 or 256.
   (a) Alternative methods for generating audio features can be found in 6_visual_features and 6-1_visual features, which uses the last hidden layer of ENV2 and a trained CNN autoencoder, respectively. The code for

training the autoencoder can be found in 6-1_visual_features.

7. 7_transformer_train/transformer_training.py: This script trains a custom dual input transformer on the audio and video input. The input audio and video features are padded, concatenated, and inputted to the transformer together for each time step (frame). The transformer itself uses a multi-head self-attention mechanism. This model predicts the desynchronization frames between the audio and video input as a single value, treating this as a regression problem.
   (a) Scripts for training older models exist in 7-3_transformer_train/ transformer_training.py (model 1), 7-LSTM (model 2), and 7-4_3DCNN/ 3DCNN_train.py (model 3 with autoencoded features).

6 Conclusion

While our approach has not been found to be especially useful, it has illustrated the importances of cleaning noisy data as much as possible and the importance of refining models to best suit the task they are being applied to. While our models have outperformed their respective no-information-rates, this has not been by much. This is likely due to a large amount of training data including many different languages and people from all around the world.

Previous attempts were shown to work on singular individuals and provided no generalization for unseen faces. Our approach is focused on improving the literature through generalization, with very few examples per person and many people in the dataset with varying nationalities, languages, and more. Since our dataset is the major difference from literature, this is why we see higher errors and less ideal results.

Our work has highlighted the importance of refining visual and audio features when working with video data. Our work also shows that generalization for this task is likely possible, but likely requires further refinement of models and features. This task is difficult and requires large amounts of resources, both in terms of data and computing power, to effectively solve.

References

[1] T. Kikuchi and Y. Ozasa. "Watch, Listen Once, and Sync: Audio-Visual Synchronization With Multi-Modal Regression CNN". In: *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. Calgary, AB, Canada, 2018, pp. 3036–3040. DOI: 10.1109/ICASSP.2018.8461853.

[2] K. Kumar et al. "Audio-visual speech synchronization detection using a bimodal linear prediction model". In: *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. Miami, FL, USA, 2009, pp. 53–59. DOI: 10.1109/CVPRW.2009.5204303.

[3] Naji Khosravan, Shervin Ardeshir, and Rohit Puri. *On Attention Modules for Audio-Visual Synchronization*. 2018. arXiv: 1812.06071 [cs.CV]. URL: https://arxiv.org/abs/1812.06071.

[4] G. Bradski. "The OpenCV Library". In: *Dr. Dobb's Journal of Software Tools* (2000).

[5] M. Tan and Q. Le. "EfficientNetV2: Smaller Models and Faster Training". In: *Proceedings of the 38th International Conference on Machine Learning*. Vol. 139. Virtual, Online, 2021, pp. 10096–10106.

[6] B. McFee et al. "librosa: Audio and Music Signal Analysis in Python". In: *Proceedings of the 14th Python in Science Conference*. Austin, TX, USA, 2015, pp. 18–24. DOI: 10.25080/Majora-7b98e3ed-003.

[7] Ashish Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[8] G. Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: *Image Analysis. SCIA 2003*. Ed. by J. Bigun and T. Gustavsson. Vol. 2749. Lecture Notes in Computer Science. Springer, Berlin, Heidelberg, 2003. DOI: 10.1007/3-540-45103-X_50. URL: https://doi.org/10.1007/3-540-45103-X_50.

[9] J. Deng et al. "RetinaFace: Single-Shot Multi-Level Face Localisation in the Wild". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2020.

[10] A. Ephrat et al. "Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation". In: *arXiv preprint arXiv:1804.03619* (2018). arXiv: 1804.03619 [cs.AI].

[11] P. Sujatha and M. R. Krishnan. "Lip feature extraction for visual speech recognition using Hidden Markov Model". In: *2012 International Conference on Computing, Communication and Applications*. Dindigul, India, 2012, pp. 1–5. DOI: `10.1109/ICCCA.2012.6179154`.

[12] Ralf C. Staudemeyer and Eric Rothstein Morris. *Understanding LSTM – a tutorial into Long Short-Term Memory Recurrent Neural Networks*. 2019. arXiv: `1909.09586 [cs.NE]`. URL: `https://arxiv.org/abs/1909.09586`.